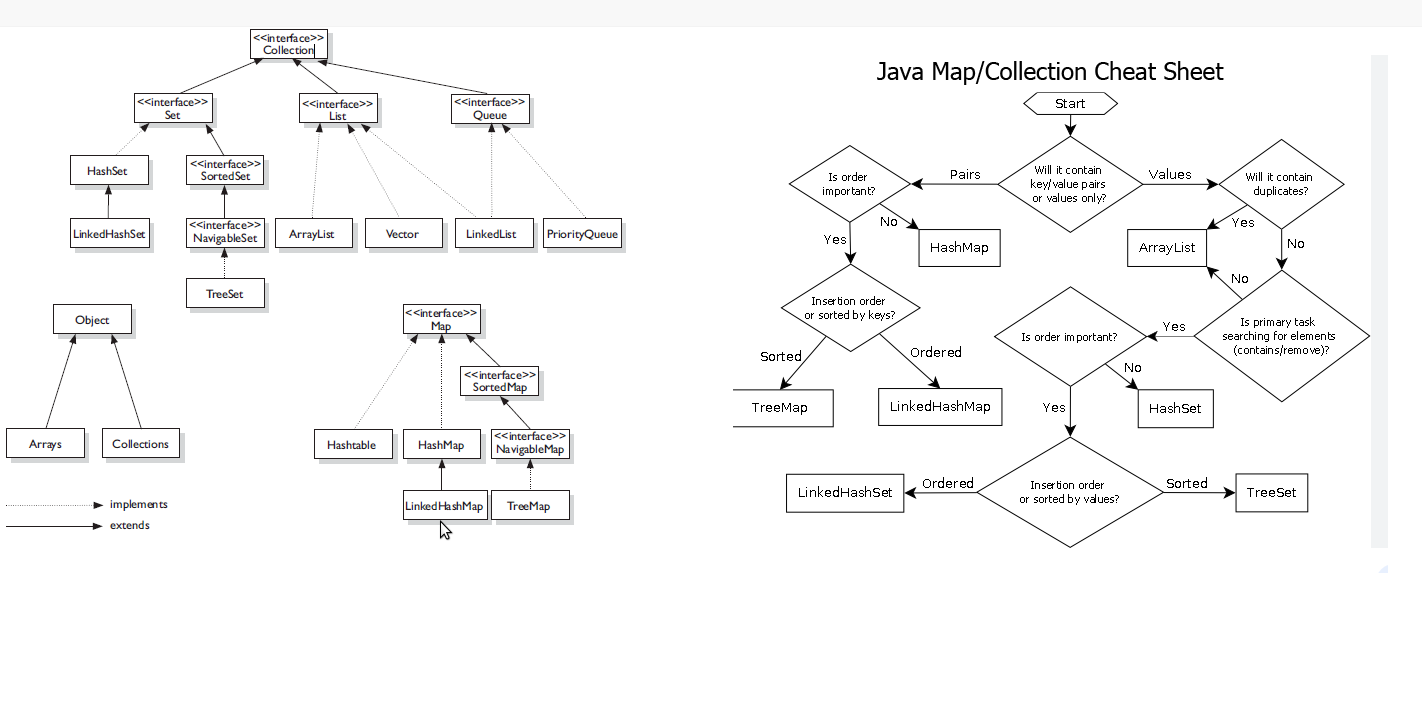
vv

**Common Points for Collections:**

1. Every Collection object already implements serializable and cloneable interfaces
2. Arraylist and vector Collection object implements Random Access Interface
3. By default, Arraylist object is non synchronized but we can get synchronized version of array list by using Collections class Synchronized List method

List🡪 public static List SynchronizedList(List l);

1. Similarly, we can get synchronized version for both Set and Map as below

Set 🡪 public static Set SynchronizedSet (Set s);

Map 🡪 public static Map SynchronizedMap (Map m);

**ArrayList**

1. Arraylist is best choice if our frequent operation is retrieval operation (Because Arraylist implements Random Access interface)
2. Arraylist is the worst choice if our frequent operation is Insertion or Deletion
3. ArrayList()
4. ArrayList(int capacity)

**LinkedList:**

1. The underlying structure is Double Linked List
2. Insertion order is Preserved
3. Duplicates are allowed
4. Heterogenous Objects are allowed
5. Null insertion is possible
6. Linked List implements Serializable and Cloneable interfaces abut not RandomAccess Interface
7. LinkedList is the best choice io our frequent operation is Insertion or deletion in middle
8. LinkedList is worst choice if our frequent operation is retrieval
9. Methods:

void addFirst()

void addLast()

void getFirst()

void getLast()

void removeFirst()

void removeLast()

1. **Constructors**:

* LinkedList l =new LinkedList ();
* LinkedList l = new LinkedList (Collection c);

**Vector:**

1. The underlying structure for the vector is Resizable and Growable array
2. Duplicates are allowed
3. Insertion Order is Preserved
4. ‘null’ insertion is possible
5. Heterogenous objects are allowed
6. Vector Class implements Serializable, cloneable and RandomAccess Interface
7. Most of the methods are threadsafe, hence vector class is synchronised
8. Best for retrieval
9. **Constructors:**

* Vector v =new Vector ();

Initial Capacity is 10

New Capacity is 2\* Initial Capacity

* + - Vector v = new Vector (int InitialCapacity);
    - Vector v = new Vector (int InitialCapacity, Int IncrementalCapacity);
    - Vector v = new Vector (Collection c);

**Stack:**

1. It is child class of Vector
2. It is specially designed for LIFO
3. **Constructors:**
   1. Stack s = new Stack();
4. **Methods**::

* push(Object 0);
* pop()
* peek()
* empty()
* search()

# **CURSORS**

If we need to retrieve the objects from Collection one by one, then we need to use Cursors

1. Enumeration(I)
2. Iterator(I)
3. ListIterator(I)

1.Enumeration: We can create Enumeration object by creating elements() method from Vector class

Public Enumeration elements();

2.We have two methods

Public Boolean hasElements();

Public object nextElement();

Syntax::

Vector v = new Vector();  
 for (int i = 0; i <= 10; i++) {  
 v.addElement(i);  
  
  
 }  
 System.*out*.println(v);  
  
 Enumeration e = v.elements();  
 while (e.hasMoreElements()) {  
 Integer i =(Integer) e.nextElement();  
 if (i % 2 == 0)  
 System.*out*.println(i);  
 }}

3.Enumeration is only applicable for legacy classes and hence it is not a universal curser

4.There is no remove operation

**Iterator::**

1. Iterator is a universal cursor
2. By using iterator, we can perform both read and remove operations
3. We can create Iterator object by using iterator() method of collection Interface

Public Iterator iterator();

Eg:

public static void main(String[] args) {  
  
 ArrayList al = new ArrayList();  
 for (int i = 0; i <= 10; i++) {  
 al.add(i);  
  
 }  
 System.*out*.println(al);  
  
 Iterator itr = al.iterator();  
 while (itr.hasNext()) {  
 Integer i = (Integer) itr.next();  
 if (i % 2 == 0)  
 System.*out*.println(i);  
 else  
 itr.remove();

**Limitations::**

1. By Using Enumeration and Iterator we can move only towards forward direction and we can’t move to the backward direction and hence these are single direction cursors
2. By using Iterator, we can perform only read and remove operations and we can’t perform replacement of new objects

List Iterator::

1. By Using listIterator we can move either to the forward direction and backward direction, hence list iterator is a bidirectional curser
2. By using ListIterator we can perform read, remove, replace and addition of objects also
3. We can create ListIterator object by using listIterator() method of List Interface

Eg: public ListIterator ListIterator ();

ListIterator itr =l.ListIterator()

Methods of ListIterator::

ListIterator is the child interface of Iterator and hence all the methods of Iterator by default available to ListIterator

public boolean hasNext();

public object next(); FORWARD DIRECTION

public int nextIndex();

public boolean hasPrevious();

public object previous(); BACKWARD DIRECTION

public int previousIndex();

public void remove();

public void set(); Other Methods

public void add();

**SET:**

1. Set is the child Interface of collection
2. If we want to represent a group of individual objects as a single entity, where duplicates are not allowed and insertion order is not preserved ,then we should go for Set
3. Set Interface does not any new methods so we need to use only Collection Interface Method

**HashSet:**

1. The underlying data structure is HashTable
2. Duplicates are not allowed, if we are trying to add any duplicates, we wont get any compile time or run time error, add() method simply returns false
3. Insertion order is not preserved and all objects are inserted based upon hashcode of objects
4. Heterogenous objects are allowed
5. ‘null’ insertion is possible
6. Implements Serializable and clonable interface but not RandomAccess Interface
7. HashSet is the best choice if our frequent operation is search operation

**Constructors:**

* HashSet hs = new HashSet();🡪 fill ratio / LoadFactor 0.75
* HashSet hs = new HashSet (int InitialCapacity); 🡪 Fill Ratio / LoadFactor 0.75
* HashSet hs = new HashSet (int Initialcapacity, Float LoadFactor);
* HashSet hs = new HashSet (Collection c);

**LinkedHashset:**

1. The underlying data structure is Hashtable+ LinkedLIst
2. Insertion Order is Preserved
3. Duplicates are not allowed
4. Same as HashSet

**SortedSet::**

1. It is child Interface of Set
2. If we want to represent a group of individual objects according to some sorting order and duplicates are not allowed then we should use SortedSet

**Methods:**

* Object first()-returns firsr element
* Object last()-returns last element
* SortedSet headset()
* SortedSet tailSet()
* SortedSet subSet()
* Comparator.comparator()

**TreeSet::**

1. The underlying data structure for TreeSet is Balanced Tree
2. Duplicates are not allowed
3. Insertion order is not preserved, but all objects are inserted according to some sorting order
4. Heterogenous objects are not allowed, we will get ClassCastException if we are trying to insert heterogenous objects
5. Null insertion is allowed but only once

**Constructors:**

TreeSet t = new TreeSet()🡪Default natural sorting Order

Treeset t =new Treeset(Comparator c)🡪 Customized Sorting Order

TreeSet t = new TreeSet(SortedSet s)

TreeSet t = new TreeSet(Collection c)

**Comparable (I):**

1. It is present in java.lang package
2. It contains only one method compareTo()

Public int compareTo();

1. If we are depending on Default natural sorting order then while adding objects into TreeSet. JVM will call compareTo() method

Eg:: obj1.compareTo(obj2)

* + returns **-ve** iff obj1 has to come before obj2
  + returns **+ve** iff ob1 has to come after obj2
  + returns **0** iff obj1 and obj2 are equal

eg: public class Comparablee {  
 public static void main(String[] args) {  
 System.*out*.println("D".compareTo("F"));  
 System.*out*.println("F".compareTo("A"));  
 System.*out*.println("A".compareTo("A"));  
 System.*out*.println("A".compareTo(null));

o/p : -2

5

0

NullPointerException

**Comparator(I):**

1. It is used for Customised Sorting order
2. It is present in java.utils package
3. It has two methods

compare()

equals()

compare() Method:

eg: public int compare(Object obj1,Object obj2);

* + returns **-ve** iff obj1 has to come before obj2
  + returns **+ve** iff ob1 has to come after obj2
  + returns **0** iff obj1 and obj2 are equal

equals() Method::

It is from Object Class so need to provide implementation for equals method

**Maps:**

1. It is Not a child interface of collection
2. If we want to represent a group of objects as key value pairs then we have to use Map
3. Map is considered a a collection of entry objects
4. Without existing of entry object there is no chance of map object

**Main Methods::**

* Set KeySet()
* Collection values () Collection views of Map
* Set KeyValues()

**EntrySet::**

Entry Interface:

Each Key Value pair is called Entry

**Methods:::**

**These 3 methods are Entry Specific methods and we can apply only Entry Objects**

Object getKey()

Object getValue()

Object setValue(Object newObj)

HashMap:

1. The underlying data Structure is HashTable
2. Insertion Order is not preserved and it is based on Hashcode of Keys
3. Duplicate Keys are not allowed, but Values can be duplicated
4. Heterogenous objects are allowed for both Key and Value
5. Null is allowed for key but only foe once
6. Null is allowed for Values for multiple times
7. HashMap Implements Serializable and cloneable Interfaces but not RandomAccess
8. HashMap is best choice if our frequent operation is search operation

Constructors:

* HashMap hm = HashMap();

Int Capacity = 16

Fill Ratio = 0.75

* HashMap hm = new HashMap(Int Initial Capacity);
* HashMap hm = new HashMap(int InitialCapacity Float FillRatio);
* HashMap hm = new HashMap(Map m);

Methods:

**LinkedHashMap:**

1. It is same as HashMap including methods and constructors
2. The underlying data structure is LinkedList and HashTable
3. Insertion order is preserved

HashTable:

1. The underlying data structure Is Hashtable data structure only
2. Insertion Order is not preserved and it is based om hashcode of keys
3. Duplicate keys are not allowed and values can be duplicated
4. Heterogenous objects are allowed for both keys and values
5. Null is not allowed for both key and values otherwise we will get run time exception as NPE
6. It implements and cloneable interfaces but not random access
7. Ievery Method present Is hashtable is synchronised and hence hashtable object is threadsafe
8. It is best choice iff four frequent operation is search operation

Constructors ::

1. Hashtable hm = new Hashtable ()

Initial Capacity =11

Fill ratio = 0.75

1. Hashtable h = new Hashtable (int InitialCapacity);
2. Hashtable h = new Hashtable (int InitialCapacity, Float fillRatio);
3. Hashtable h =new Hashtable (Map m);

SortedMap::

1. It is child interface of Map
2. If we want to represent a group of value according to some sorting order of keys,then we should go for SortedMap

**Sorted map defines the following specific methods**

1. Object firstKey()
2. Object lastKey()
3. SortedMap headMap(Object key)
4. sortedMap tailMap(Object key)
5. SortedMap submap(Object obj1,Object obj2)
6. Comparater comparator()

**TreeMap:**

1. The underlying data structure is Red Black Tree
2. Insertion order is not preserved and it is based on some sorting order of Keys
3. Duplicate keys are not allowed but values can be duplicated
4. If we are defining on default sorting order then keys should be Homogenous and comparable otherwise, we will get Run Time Exception saying ClassCastExeception
5. If we are our own sorting by comparator then keys need not be homogenous and comparable, we can take heterogenous and non-comparable objects also
6. Whether we are depending on default natural sorting order or customized sorting order there are no restrictions for values

**Constructers:**

TreeMap t = new TreeMap();🡪Default natural sorting Order  
TreeMap t =new TreeMap(Comparator c);🡪 Customized Sorting Order  
TreeMap t = new TreeMap(Map m);  
TreeMap t = new TreeMap(SortedMap s);

Properties:

* String setProperty(String pname,String pvalue)
* String getProperty(String pname)
* Enumeration propertyNames() :🡪 returns all property names present In properties object
* Void load(InputStream is) 🡪 to load properties file into java properties object
* Void store(OutputStream os, String Comment 🡪 to Store properties from java properties object into properties file